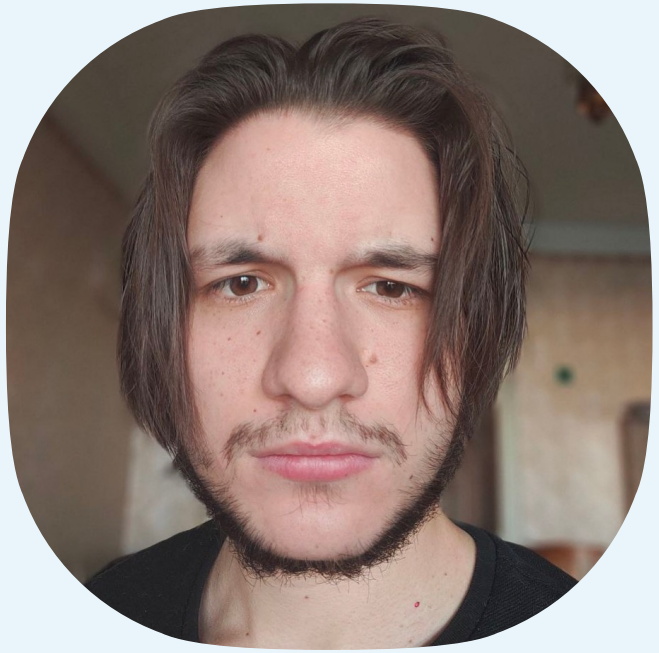
tech

GoFunc 2024

Строки от известного к неизвестному

Чалый Роман
Go developer VK Tech

бизнес



Чалый Роман

Go developer

На Go пишу с 2017

Люблю копаться в коде

tech

Будет ли тут аллокация?

```go
var a, b string
...
c := a + b
```

Будет ли тут аллокация?

```go
var a, b string
...
c := a + b
```

```go
func concatstring2(*tmpBuf, string, string) string
func concatstring3(*tmpBuf, string, string, string) string
func concatstring4(*tmpBuf, string, string, string, string) string
func concatstring5(*tmpBuf, string, string, string, string, string) string
```

Agenda

1. **Кратко про про строки**
   Что это такое и немного про UTF-8
2. **Кастинг слайс байт в строку**
   Как работает, какие бывают оптимизации
3. **Кастинг строки в слайс байт**
   Как работает, какие бывают оптимизации
4. **Конкатенация строк**
   Почему так много функций для конкатенации
5. **Итог**
   Как стоит подойти к оптимизациям

tech

Кратко
про строки

#строка

Строка — это read only массив байт

```
v := "test"
v[0] = ‘n’
// Так нельзя :)
```

tech

#строка

Строка — это read only массив байт

```
v := "test"
v[0] = ‘n’
// Так нельзя :)
```

Копируется при любом изменении

tech

#строка

Строка — это read only массив байт

```
v := "test"
v[0] = ‘n’
// Так нельзя :)
```

Копируется при любом изменении

Создается новый объект при кастинге в `[]byte` и в `[]rune`

tech

ИМХО

1

Конкурентность
Меньше шансов ошибиться
при параллельном доступе

2

Строка это UTF-8
Сложно работать на уровне байтов

ИМХО

1. **Конкурентность**
   Меньше шансов ошибиться при параллельном доступе

2. **Строка это UTF-8**
   Сложно работать на уровне байтов

#строка

UTF-8

```
// [01101000 01101001]
fmt.Printf("%08b\n", []byte("hi"))

// [01101000 01101001 11010000 10111111]
fmt.Printf("%08b\n", []byte("hiп"))

// [01101000 01101001 11110000 10011111 10010001 10001011]
fmt.Printf("%08b\n", []byte("hi👋"))
```

tech

#строка

UTF-8

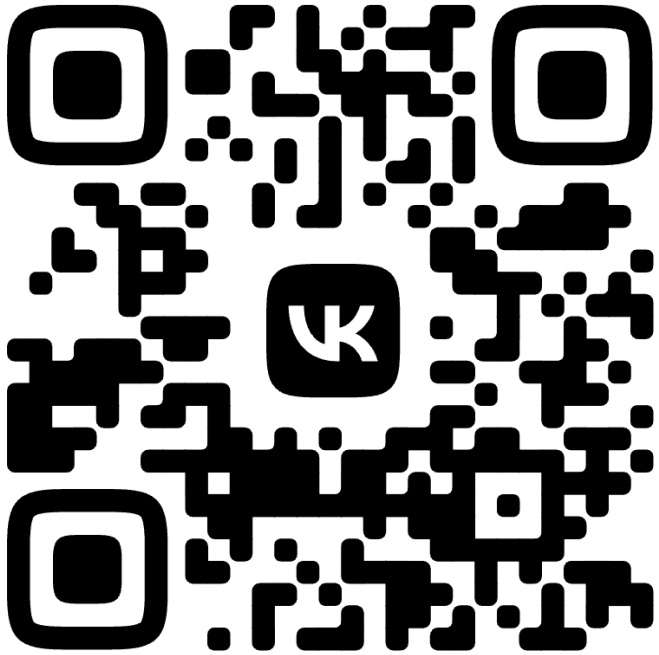
Типичная ошибка

```go
const s = "hi👋"

// count of bytes: 6
fmt.Println("count of bytes:", len(s))


// count of code point 3
fmt.Println(
    "count of character:",
    utf8.RuneCountInString(s),
)

// Code point != напечатанный символ
```

tech

#строка

Операции, которые вроде как меняют строку



```
name                time/op
BytesToString-10     133ns ± 6%
StringToBytes-10     141ns ± 6%
Concat-10            234ns ± 5%

name                alloc/op
BytesToString-10    1.15kB ± 0%
StringToBytes-10    1.15kB ± 0%
Concat-10           2.30kB ± 0%

name                allocs/op
BytesToString-10      1.00 ± 0%
StringToBytes-10      1.00 ± 0%
Concat-10             1.00 ± 0%
```

tech

Кастинг слайс байт в строку

Немного про компиляцию

1

Go code

Код, который
мы пишем

2

«PseudoAsm»

Абстракция внутри
Go, основан
на OS Plan 9 asm

3

Asm

Инструкции
целевой машины

Через переменную

```go
func bytesToString(v []byte) string {
    return string(v)
}
```

```go
func slicebytetostring(buf *tmpBuf, ptr *byte, n int) string {
    var p unsafe.Pointer
    if buf != nil && n <= len(buf) {
        p = unsafe.Pointer(buf)
    } else {
        p = mallocgc(uintptr(n), nil, false)
    }
    memmove(p, unsafe.Pointer(ptr), uintptr(n))
    return unsafe.String((*byte)(p), n)
}
```

#bytestostring

Через переменную

```go
func bytesToString(v []byte) string {
    return string(v)
}
```

```go
func slicebytetostring(buf *tmpBuf, ptr *byte, n int) string {
    var p unsafe.Pointer
    if buf != nil && n <= len(buf) {
        p = unsafe.Pointer(buf)
    } else {
        p = mallocgc(uintptr(n), nil, false)
    }
    memmove(p, unsafe.Pointer(ptr), uintptr(n))
    return unsafe.String((*byte)(p), n)
}
```

Оптимизации

Used for `m[string(k)]`

Мы не можем работать с полученной строкой

Used for `"<"+string(b)+">"`

Мы не можем работать с полученной строкой

Used for `string(b) == "foo" comparison`

Мы не можем работать с полученной строкой

#bytestostring

```
// ConcatDiffOp
s := string(someBytes)
v := "test" + s + "test”
if v == "" {
    b.Fatal()
}
```

```
// ConcatSameOp
v := "test" +
    string(someBytes) +
    "test”
if v == "" {
    b.Fatal()
}
```

#bytestostring

Как сломать оптимизацию

```
// ConcatDiffOp
s := string(someBytes)
v := "test" + s + "test”
if v == "" {
    b.Fatal()
}
```

```
// ConcatSameOp
v := "test" +
    string(someBytes) +
    "test”
if v == "" {
    b.Fatal()
}
```

```
Name                    time/op
ConcatSameOp10          41.0ns ± 1%
ConcatDiffOp-10         65.0ns ± 3%

name                    alloc/op
ConcatSameOp-10         128B ± 0%
ConcatDiffOp-10         240B ± 0%

name                    allocs/op
ConcatSameOp-10         1.00 ± 0%
ConcatDiffOp-10         2.00 ± 0%
```

Черная магия

```
func bytesToString(b []byte) string {
   return *(*string)(unsafe.Pointer(&b))
}
```

Не задокументировано, но разрешено в соответсвии с unsafe rule (1)

Черная магия

```go
func bytesToString(b []byte) string {
   return *(*string)(unsafe.Pointer(&b))
}
```

Не задокументировано, но разрешено в соответсвии с unsafe rule (1)

```go
type slice struct {
    array unsafe.Pointer
    len   int
    cap   int
}
```

```go
type stringStruct struct {
    str unsafe.Pointer
    len int
}
```

Черная магия

```go
func bytesToString(b []byte) string {
   return *(*string)(unsafe.Pointer(&b))
}
```

Не задокументировано, но разрешено в соответсвии с unsafe rule (1)

```go
type slice struct {
    array unsafe.Pointer
    len   int
    cap   int
}
```

```go
type stringStruct struct {
    str unsafe.Pointer
    len int
}
```

Черная магия

```go
func bytesToString(b []byte) string {
   return *(*string)(unsafe.Pointer(&b))
}
```

Не задокументировано,но разрешено в соответсвии с unsafe rule (1)

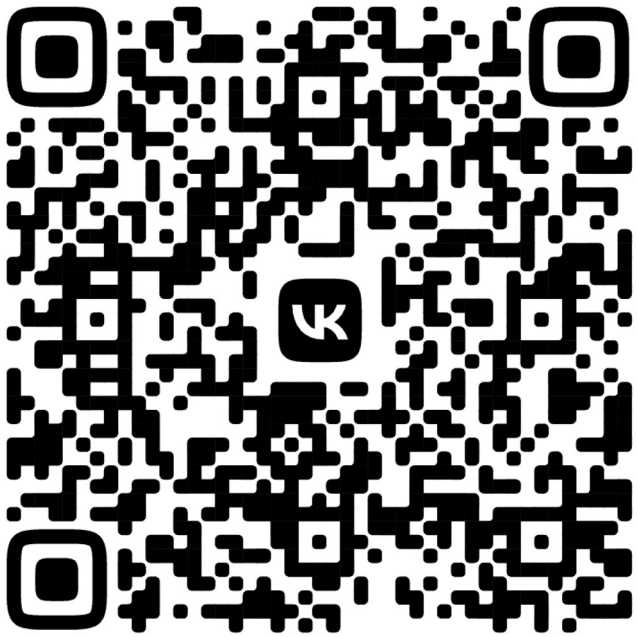
```go
type slice struct {
    array unsafe.Pointer
    len   int
    cap   int
}
```

```go
type stringStruct struct {
    str unsafe.Pointer
    len int
}
```

>1.20

```go
func bytesToString(v []byte) string {
    return unsafe.String(unsafe.SliceData(v), len(v))
}
```

Buffer?



Коммит с добавлением буфера

```
// The constant is known to the compiler.
// There is no fundamental theory behind this
// number.
const tmpStringBufSize = 32

type tmpBuf [tmpStringBufSize]byte
```

Кастинг строки в слайс байт

Constant

```go
func stringToByte() []byte {
    return []byte("hello world")
}
```

```
LEAQ    type:[11]uint8(SB), AX
CALL    runtime.newobject(SB)
MOVQ    $8031924123371070824, CX
MOVL    $1684828783, 7(AX)
MOVL    $11, BX
```

Constant

```go
func stringToByte() []byte {
    return []byte("hello world")
}
```

```
LEAQ    type:[11]uint8(SB), AX
CALL    runtime.newobject(SB)
MOVQ    $8031924123371070824, CX
MOVL    $1684828783, 7(AX)
MOVL    $11, BX
```

Constant

```go
func stringToByte() []byte {
    return []byte("hello world")
}
```

```
LEAQ    type:[11]uint8(SB), AX
CALL    runtime.newobject(SB)
MOVQ    $8031924123371070824, CX
MOVL    $1684828783, 7(AX)
MOVL    $11, BX
```

Constant

```go
func stringToByte() []byte {
    return []byte("hello world")
}
```

```
LEAQ    type:[11]uint8(SB), AX
CALL    runtime.newobject(SB)
MOVQ    $8031924123371070824, CX
MOVL    $1684828783, 7(AX)
MOVL    $11, BX
```

```go
func stringToByte() []byte {
    return []byte("e")
}
```

```
LEAQ    type:[1]uint8(SB), AX
CALL    runtime.newobject(SB)
MOVB    $101, (AX)
MOVL    $1, BX
```

Variable

```go
func stringToByte(v string) []byte {
    return []byte(v)
}
```

```go
func stringtoslicebyte(buf *tmpBuf, s string) []byte {
    var b []byte
    if buf != nil && len(s) <= len(buf) {
       *buf = tmpBuf{}
        b = buf[:len(s)]
    } else {
        b = rawbyteslice(len(s))
    }
    copy(b, s)
    return b
}
```

Черная магия

```
func f(b string) []byte {
   return *(*[]byte)(unsafe.Pointer(&sl))
}
```

Не разрешено в соответствии с unsafe rule (1)

Черная магия

```go
func f(b string) []byte {
   return *(*[]byte)(unsafe.Pointer(&sl))
}
```

Не разрешено в соответствии с unsafe rule (1)

```go
sl := strings.Repeat("a", 1024)
v := *(*[]byte)(unsafe.Pointer(&sl))
// cap: 1374389856056 len: 1024
fmt.Println("cap:", cap(v), "len:", len(v))
```

Capacity будет псевдослучайным

Черная магия

```go
func f(b string) []byte {
   return *(*[]byte)(unsafe.Pointer(&sl))
}
```

Не разрешено в соответствии с unsafe rule (1)

```go
type stringStruct struct {
    str unsafe.Pointer
    len int
}
```

```go
type slice struct {
    array unsafe.Pointer
    len   int
    cap   int
}
```

Оптимизации

for I, b := range []byte(someString)

Мы не можем работать с полученным слайсом

????

>= Go 1.22

1

Escape анализ смог
помечать read only память
`attrMutates`

>= Go 1.22

1

Escape анализ смог
помечать read only память
`attrMutates`

2

Если полученный слайс не меняется,
то копировать его не надо

#bytestostring

Немного кода с компилятора

```
if n.Op() == ir.OSTR2BYTES &&
    !loc.hasAttr(attrMutates) {
    n.SetOp(ir.OSTR2BYTESTMP)
}
```

Оптимизации

`for i, b := range []byte(someString)`

Мы не можем работать с полученным слайсом

Если полученный слайс не будет меняться

Конкатенация строк

#concat

А с конкатенацией что?

```go
var cat string
cat += " /\\_/\\\n"
cat += "( o.o )\n"
cat += " > ^ <\n"
fmt.Println(cat)

// Output:
// /\_/\
//( o.o )
// > ^ <
```

tech

Constant

```go
const prefix = "hello "

func concatWorld() string  {
    return prefix + "world"
}

func concatGofunc() string  {
    return prefix + "gofunc"
}
```

```
…
LEAQ    go:string."hello world"(SB), AX
…
```

```
…
LEAQ    go:string."hello gofunc"(SB), AX
…
```

#concat

Variable

```
s, b := rawstringtmp(buf, l)
for _, x := range a {
    copy(b, x)
    b = b[len(x):]
}
return s
```

Будет ли тут аллокация?

```go
var a, b string
...
c := a + b
```

#concat

Нужно больше функций

```
func concatstring2(*tmpBuf, string, string) string

func concatstring3(*tmpBuf, string, string, string) string

func concatstring4(*tmpBuf, string, string, string, string) string

func concatstring5(*tmpBuf, string, string, string, string, string) string

// >5 (a0 + a1 + a2 + a3 + a4 + a5)
func concatstrings(*tmpBuf, []string) string
```

concatstringN

```
func concatstring2(buf *tmpBuf, a0, a1 string) string {
    return concatstrings(buf, []string{a0, a1})
}
```

#concat

Откуда `concatstring{2,3,4,5}`?

#concat

Откуда `concatstring{2,3,4,5}`?

2013

Оптимизация Си varargs
runtime Go еще на Си

#concat

Откуда concatstring{2,3,4,5}?

2013

Оптимизация Си varargs
runtime Go еще на Си

Перенесено как есть из
Си в Go

2014

#concat

Откуда concatstring{2,3,4,5}?

2013

Оптимизация Си varargs
runtime Go еще на Си

Перенесено как есть из
Си в Go

2014

2018

Quasilyte (Искандер
Шарипов) предложил
оптимизацию для
concatstring2

#concat

Откуда concatstring{2,3,4,5}?

2013

Оптимизация Си varargs runtime Go еще на Си

2014

Перенесено как есть из Си в Go

2018

Quasilyte (Искандер Шарипов) предложил оптимизацию для concatstring2

2024

Я предложил избавиться от concatstring{2,3,4,5}

Итог

Итог

```
for i, v := range []byte(str)
     map[string(bytes)]
            etc…
```

Оптимизации неочевидны

+

+

=

Итог

```
for i, v := range []byte(str)
     map[string(bytes)]
            etc…
```

Оптимизации неочевидны

+

```
zero copy for
string -> []byte
```

Добавляются новые оптимизации

+

=

Итог

```
for i, v := range []byte(str)
map[string(bytes)]
etc…
```

Оптимизации неочевидны

+

```
zero copy for
string -> []byte
```

Добавляются новые оптимизации

+

```
wrong capacity
string -> []byte
with unsafe
```

Наши оптимизации могут сломать код

=

Итог

```
for i, v := range []byte(str)
map[string(bytes)]
etc…
```

Оптимизации неочевидны

+

```
zero copy for
string -> []byte
```

Добавляются новые оптимизации

+

```
wrong capacity
string -> []byte
with unsafe
```

Наши оптимизации могут сломать код

=

Лучше выделить время на изучение оптимизаций, которые предоставляет компилятор, и использовать их, а не писать свои

Инструментарий

Git blame
Просмотр историй изменений

Godbolt
В браузере можно сравнивать ASM go кода

Source code of go
Просмотр самого исходного кода компилятора и рантайма

tech

Спасибо за внимание

tech.vk.com

бизнес